

Teoría de Lenguajes

2dos Parciales (y recuperatorios) Resueltos

Sebastián Taboh

22 de junio de 2018



Este apunte contiene soluciones a segundos parciales y recuperatorios de la materia Teoría de Lenguajes.

Este documento fue creado cuando cursé en el 2017, en caso de encontrar errores, por favor comunicarlo por mail a sebi.282@hotmail.com.

Índice

2015 C2 Recuperatorio	3
Ejercicio 1	3
Ejercicio 2	6
2017 C1 Parcial	8
Ejercicio 1	9
Ejercicio 3	14
2017 C2 Parcial	18
Ejercicio 2	19
Ejercicio 3	23

2015 C2 Recuperatorio

Ejercicio 1

(30 pts) La siguiente gramática representa un fragmento de las expresiones de la lógica proposicional: $G_1 = \langle \{E\}, \{p, \rightarrow, \vee, \neg, (,)\}, E, P \rangle$, con P :

$$E \longrightarrow E \rightarrow E \mid E \vee E \mid \neg E \mid p \mid (E)$$

(a) Dar su tabla SLR, señalando todos los conflictos que tenga.

Primero aumentamos la gramática. Queda $G'_1 = \langle \{E', E\}, \{p, \rightarrow, \vee, \neg, (,)\}, E', P' \rangle$, con P' :

$$\begin{aligned} E' &\longrightarrow E \\ E &\longrightarrow E \rightarrow E \mid E \vee E \mid \neg E \mid p \mid (E) \end{aligned}$$

Ahora armamos el autómata con los conjuntos de ítems LR(0).

Estado 0:

$$\begin{aligned} E' &\longrightarrow \cdot E \\ E &\longrightarrow \cdot E \rightarrow E \\ E &\longrightarrow \cdot E \vee E \\ E &\longrightarrow \cdot \neg E \\ E &\longrightarrow \cdot p \\ E &\longrightarrow \cdot (E) \end{aligned}$$

Estado 1:

$$\begin{aligned} E' &\longrightarrow E \cdot \\ E &\longrightarrow E \cdot \rightarrow E \\ E &\longrightarrow E \cdot \vee E \end{aligned}$$

Estado 2:

$$\begin{aligned} E &\longrightarrow \neg \cdot E \\ &\quad - - - - \\ E &\longrightarrow \cdot E \rightarrow E \\ E &\longrightarrow \cdot E \vee E \\ E &\longrightarrow \cdot \neg E \\ E &\longrightarrow \cdot p \\ E &\longrightarrow \cdot (E) \end{aligned}$$

Estado 3:

$$E \longrightarrow p \cdot$$

Estado 4:

$$\begin{array}{c}
 E \longrightarrow (\cdot E) \\
 \text{---} \text{---} \text{---} \text{---} \\
 E \longrightarrow \cdot E \rightarrow E \\
 E \longrightarrow \cdot E \vee E \\
 E \longrightarrow \cdot \neg E \\
 E \longrightarrow \cdot p \\
 E \longrightarrow \cdot (E)
 \end{array}$$

Estado 5:

$$\begin{array}{c}
 E \longrightarrow E \rightarrow \cdot E \\
 \text{---} \text{---} \text{---} \text{---} \\
 E \longrightarrow \cdot E \rightarrow E \\
 E \longrightarrow \cdot E \vee E \\
 E \longrightarrow \cdot \neg E \\
 E \longrightarrow \cdot p \\
 E \longrightarrow \cdot (E)
 \end{array}$$

Estado 6:

$$\begin{array}{c}
 E \longrightarrow E \vee \cdot E \\
 \text{---} \text{---} \text{---} \text{---} \\
 E \longrightarrow \cdot E \rightarrow E \\
 E \longrightarrow \cdot E \vee E \\
 E \longrightarrow \cdot \neg E \\
 E \longrightarrow \cdot p \\
 E \longrightarrow \cdot (E)
 \end{array}$$

Estado 7:

$$\begin{array}{c}
 E \longrightarrow \neg E \cdot \\
 E \longrightarrow E \cdot \rightarrow E \\
 E \longrightarrow E \cdot \vee E
 \end{array}$$

Estado 8:

$$\begin{array}{c}
 E \longrightarrow (E \cdot) \\
 E \longrightarrow E \cdot \rightarrow E \\
 E \longrightarrow E \cdot \vee E
 \end{array}$$

Estado 9:

$$\begin{array}{c}
 E \longrightarrow E \rightarrow E \cdot \\
 E \longrightarrow E \cdot \rightarrow E \\
 E \longrightarrow E \cdot \vee E
 \end{array}$$

Estado 10:

$$E \longrightarrow E \vee E \cdot$$

$$E \longrightarrow E \cdot \rightarrow E$$

$$E \longrightarrow E \cdot \vee E$$

Estado 11:

$$E \longrightarrow (E) \cdot$$

Recordemos que SLR(1) sólo reduce para los símbolos en SIGUIENTES de ese no terminal. Así, hay que calcular SIGUIENTES para cada no terminal.

NT	SIGUIENTES
E'	$\{\$ \}$
E	$\{\rightarrow, \$, \vee,)\}$

ESTADO	ACTION							GoTo
	p	\rightarrow	\vee	\neg	$($	$)$	$\$$	E
0	s3			s2	s4			1
1		s5	s6				accept	
2	s3			s2	s4			7
3		r(4)	r(4)			r(4)	r(4)	
4	s3			s2	s4			8
5	s3			s2	s4			9
6	s3			s2	s4			10
7		s5/r(3)	s6/r(3)			r(3)	r(3)	
8		s5	s6			s11		
9		s5/r(1)	s6/r(1)			r(1)	r(1)	
10		s5/r(2)	s6/r(2)			r(2)	r(2)	
11		r(5)	r(5)			r(5)	r(5)	

$$r(1) = r(E \longrightarrow E \rightarrow E)$$

$$r(2) = r(E \longrightarrow E \vee E)$$

$$r(3) = r(E \longrightarrow \neg E)$$

$$r(4) = r(E \longrightarrow p)$$

$$r(5) = r(E \longrightarrow (E))$$

- (b) Resolver los conflictos eligiendo en cada caso una de las entradas de la gramática de manera que el árbol de derivación resultante respete las siguientes reglas de precedencia y asociatividad: la negación tiene mayor precedencia que la disyunción, que es asociativa a izquierda y tiene mayor precedencia que la implicación, que es asociativa a derecha.

Si la negación tiene mayor precedencia que la disyunción significa que si se puede reducir $\neq E$ a E o shiftear por un \vee , la reducción de la negación se tiene que dar antes que el shifteo. Es decir, tiene que ocurrir la primera. Esto resuelve el conflicto de $(7, \vee)$.

El de $(10, \vee)$ se resuelve porque la disyunción es asociativa a izquierda, o sea que tiene más prioridad reducir $E \vee E$ a E que shiftear por leer otro \vee .

La mayor precedencia de la disyunción sobre la implicación indica que si se puede reducir $E \vee E$ a E o shiftear por leer \rightarrow , debe darse la primera: resuelve el conflicto $(10, \rightarrow)$. También establece que si se puede reducir $E \rightarrow E$ a E o shiftear por leer \vee debe darse la segunda, lo que define la entrada de $(9, \vee)$.

La precedencia a derecha de \rightarrow nos dice cómo resolver el conflicto de $(9, \rightarrow)$.

Además, por transitividad la negación tiene mayor precedencia que la implicación, por lo que se resuelve el conflicto $(7, \rightarrow)$.

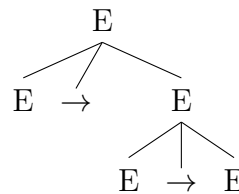
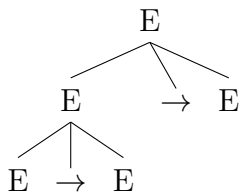
ESTADO	ACTION							GoTo
	p	\rightarrow	\vee	\neg	$($	$)$	$\$$	E
0	s3			s2	s4			1
1		s5	s6				accept	
2	s3			s2	s4			7
3		r(4)	r(4)			r(4)	r(4)	
4	s3			s2	s4			8
5	s3			s2	s4			9
6	s3			s2	s4			10
7		r(3)	r(3)			r(3)	r(3)	
8		s5	s6			s11		
9		s5	s6			r(1)	r(1)	
10		r(2)	r(2)			r(2)	r(2)	
11		r(5)	r(5)			r(5)	r(5)	

Ejercicio 2

(35 pts) Dar una gramática extendida que genere $L(G_1)$ y sea ELL(1).

Dar su parser iterativo-recursivo.

Primero notemos que esta gramática es ambigua dado que existe una cadena con más de un árbol de derivación. Por ejemplo, $p \rightarrow p \rightarrow p$.



Es importante recordar que eliminar la recursión o factorizar NUNCA van a desambiguar una gramática.

Por otra parte, dado que lo que piden es una gramática extendida, tampoco es útil usar la técnica de eliminación de la recursión a izquierda. Lo que hay que hacer es remplazar la recursión por iteración.

Entonces, lo primero es desambiguar la gramática. Podemos aprovechar que en el Ejercicio 1 nos dieron reglas de precedencia y asociatividad, aunque no es obligatorio seguirlas para este ejercicio.

Siguiendo esas reglas una gramática no ambigua posible es:

$$\begin{aligned} E &\longrightarrow D \rightarrow E \mid D \\ D &\longrightarrow D \vee A \mid A \\ A &\longrightarrow \neg A \mid p \mid (E) \end{aligned}$$

Ahora que tenemos una gramática no ambigua podemos convertirla en extendida ELL(1).
Para E podemos factorizar:

$$E \longrightarrow D(\rightarrow E)?$$

y para D convertir recursión en iteración:

$$D \longrightarrow A(\vee A)^*$$

A queda como está.

Finalmente obtenemos

```

Proc Main()
    E();
    match('$');
    accept();
End

Proc E()
    D();
    if(tc == '->'){
        match('->');
        E();
    }
End

Proc D()
    A();
    while(ct == 'v') {
        match('v');
        A();
    }
End

Proc A()
    if(ct == '!') {
        match('!');
        A();
    }
    elseif(ct == 'p'){
        match('p');
    }
    elseif(ct == '('){
        match('(');
        E();
        match(')');
    }
    else{
        error();
    }
End

```

2017 C1 Parcial

Teoría de Lenguajes - Segundo Parcial

Primer cuatrimestre de 2017

Hacer cada ejercicio en hojas separadas.

Poner nombre, número de orden y número de página en cada ejercicio.

Justificar todas las respuestas.

El examen es a libro (no electrónico) abierto.

Se aprueba con 65 puntos.

1. (30 pts) Sea G_1 una gramática libre de contexto extendida,

 $G_1 = \langle \{S, C, D\}, \{a, b, c, d, \cdot\}, S, P_1 \rangle$, con P_1 :

$$\begin{aligned} S &\rightarrow (abC \mid acD)^* a \\ C &\rightarrow c(\cdot, c)^* \\ D &\rightarrow d(d?)^* \end{aligned}$$

- a) ¿Es G_1 ELL(1)? Si no, dar una gramática G'_1 que sí lo sea, tal que $\mathcal{L}(G'_1) = \mathcal{L}(G_1)$.
b) Dar el parser recursivo-iterativo para la gramática ELL(1).

2. (40 pts) Dada la gramática G_2 , en la que se definen algunas operaciones con listas, se desea realizar una traducción dirigida por sintaxis que imprima la evaluación de la expresión dada.

 $G_2 = \langle \{S, L, L', L'', C, C', P, V\}, \{\text{function}, (\cdot), [,], \{, \}, \cdot, \text{var}, \text{num}\}, S, P_2 \rangle$, con P_2 :

$$\begin{aligned} S &\rightarrow \text{function}(L) \mid L \\ L &\rightarrow C \mid L' \\ L' &\rightarrow L'' \mid \lambda \\ L'' &\rightarrow V, L'' \mid V \\ C &\rightarrow \{C'\} \mid \lambda \\ C' &\rightarrow P, C' \mid P \\ P &\rightarrow (\text{var}, \text{num}) \\ V &\rightarrow \text{num} \mid \text{var} \mid (\text{num}, \text{num}) \end{aligned}$$

El token `function` tiene un atributo `name`, que indica la función a evaluar:

- `compress`: que, dada una lista, obtiene la versión comprimida de la misma.
Por ejemplo, `compress([1,3,3,5,5,5,3])` \rightsquigarrow `[1,(3,2),(5,3),3]`,
- `expand`: que, dada una lista, obtiene la versión expandida de la misma.
Por ejemplo, `expand([(2,8),3,10])` \rightsquigarrow `[2,2,2,2,2,2,2,2,3,10]`,
- `max`: que obtiene el máximo valor¹ de una lista, comprimida o no.
Por ejemplo, `max([9,10,-1])` \rightsquigarrow `10`, `max([9, 13, (14,10)])` \rightsquigarrow `14`

Las listas pueden contener variables, por lo que hay un contexto de variables, opcional, donde constan variables y su valor. Todas las variables deben tener un valor asignado para que se pueda evaluar una expresión.

Por ejemplo: `{(x,3),(y,1)}[3,x,34,x,5]` \rightsquigarrow `[3,3,34,3,5]`, `{(z,3)}[]` \rightsquigarrow `[]`,
`compress({(y,8),(x,8),(z,3)}[x,y,z])` \rightsquigarrow `[(8,2),3]`,
`{(x,3)}[x,x,y]` \rightsquigarrow `ERROR ('y' sin definir)`,
`max({(x,1)}[8,z])` \rightsquigarrow `ERROR ('z' sin definir)`.

3. (30 pts) Sea $G_3 = \langle \{D, R, T, V\}, \{\text{vname}, \cdot, =, \text{int}, \text{string}, \text{bool}, \text{literal}\}, D, P_3 \rangle$, con P_3 :

$$\begin{aligned} D &\rightarrow T \text{ vname } R; \\ R &\rightarrow = V \mid R R \mid \lambda \\ T &\rightarrow \text{int} \mid \text{string} \mid \text{bool} \\ V &\rightarrow \text{vname} \mid \text{literal} \end{aligned}$$

- a) Dar la tabla SLR correspondiente.
b) ¿Es G_3 SLR? Si no, indicar cómo podría/n resolverse el/los conflictos existentes sin modificar la gramática, en caso de ser posible, y justificar.

¹Si la lista está vacía, el máximo es null

Ejercicio 1

$G_1 = \text{GLC extendida.}$

$$G_1 = \langle \{S, C, D\}, \{a, b, c, d\}, S, P_1 \rangle$$

$$P_1: S \rightarrow (abC \mid acD)^* a$$

$$C \rightarrow c(c)^*$$

$$D \rightarrow d(d)^*$$

- a) Para ver si G_1 es ELL(1) vamos a ver si su derivada es LL(1).

Su derivada es:

$$S \rightarrow A_1 a$$

$$A_1 \rightarrow A_2 A_1 \mid \lambda$$

$$A_2 \rightarrow A_3 \mid A_4$$

$$A_3 \rightarrow abC$$

$$C \rightarrow cA_5$$

$$A_5 \rightarrow cA_5 \mid \lambda$$

$$A_4 \rightarrow acD$$

$$D \rightarrow dA_6$$

$$A_6 \rightarrow A_7 A_6 \mid \lambda$$

$$A_7 \rightarrow d \mid \lambda$$

$$SD(A_7 \rightarrow \lambda) = \text{Sigvientes}(A_7)$$

$$= \text{Primeros}(A_6) \cup \text{Sigvientes}(A_6)$$

$$= \text{Primeros}(A_2) \cup \text{Sigvientes}(D)$$

$$= \{d\} \cup \text{Sigvientes}(A_4)$$

$$= \{d\} \cup \text{sigvientes}(A_2)$$

$$= \{d\} \cup (\text{Primeros}(A_1) \cup \text{Sigvientes}(A_1))$$

$$= \{d\} \cup (\text{Primeros}(A_2) \cup \{a\})$$

$$= \{d\} \cup (\text{Primeros}(A_3) \cup \text{Primeros}(A_4)) \cup \{a\}$$

$$= \{d\} \cup \{a\}$$

$$SD(A_7 \rightarrow d) = \{d\}.$$

No es LL(1) ya que

$$SD(A_7 \rightarrow \lambda) \cap SD(A_7 \rightarrow d) = \{d\}.$$

Así, G_1 no es ELL(1).

Vamos a probar con la gramática

$$G_1' = \langle V_N(G_1), V_T(G_1), S, P_1' \rangle.$$

con P_1' :

$$\begin{aligned} S &\rightarrow (a(bc|cd))^* a. \\ C &\rightarrow c|c^* \\ D &\rightarrow d|d^* \end{aligned}$$

Es claro que $L(G_1) = L(G_1')$

En la producción de S sólo se sacó a de factor común y en D se usó que concatenar con λ no cambia los cadenas.

$$\begin{aligned} S &\rightarrow A_1 a \\ A_1 &\rightarrow A_2 A_1 | \lambda \\ A_2 &\rightarrow a(bc|cd) \end{aligned}$$

Esto ya parece ser conflictivo:

$$\begin{aligned} SD(A_1 \rightarrow \lambda) &= \text{Siguientes}(A_1) \ni \{a\} \\ SD(A_1 \rightarrow A_2 A_1) &= \text{Primeros}(A_2) \ni \{a\}, \end{aligned}$$

Repensemos G_1' cambiando P_1' sin cambiar el lenguaje generado.

$$\Rightarrow S \rightarrow (a(bc|c^*|cd^*))^* a.$$

$$\Rightarrow S \rightarrow (a(b|\lambda)c(|c^*|dd^*))^* a.$$

Usando un "truco" parecido al que se podía usar en el Ej. 12) b) de la P.8 llego a

$$\Rightarrow S \rightarrow a \left((b|\lambda) c (c|d^* | d d^*) a \right)^*$$

Ahora, su derivada es

$$S \rightarrow a A_1$$

$$A_1 \rightarrow A_2 A_1 | \lambda$$

$$A_2 \rightarrow A_3 c A_4$$

$$A_3 \rightarrow b | \lambda$$

$$A_4 \rightarrow A_5 | A_6$$

$$A_5 \rightarrow c A_5 | \lambda$$

$$A_6 \rightarrow d A_7$$

$$A_7 \rightarrow d A_7 | \lambda$$

$$\begin{aligned} SD(A_7 \rightarrow \lambda) &= \text{Sigvientes}(A_7) \\ &= \text{Sigvientes}(A_6) \\ &= \text{Sigvientes}(A_4) \\ &= \text{Sigvientes}(A_2) \\ &= (\text{Primeros}(A_1) \cup \text{Sigvientes}(A_1)) \\ &= \text{Primeros}(A_2) \cup \text{Sigvientes}(S) \\ &= (\text{Primeros}(A_3) \cup \text{Sigvientes}(A_3)) \cup \{\$ \} \\ &= \{b\} \cup \{c\} \cup \{\$ \}. \end{aligned}$$

$$SD(A_7 \rightarrow d A_7) = \{d\}.$$

$$\begin{aligned} SD(A_5 \rightarrow \lambda) &= \text{Sigvientes}(A_5) \\ &= \text{Sigvientes}(A_4) \\ &= \text{Sigvientes}(A_2) \\ &= \{b\} \cup \{c\} \cup \{\$ \}. \end{aligned}$$

$$SD(A_5 \rightarrow c A_5) = \{c\}.$$

$$SD(A_4 \rightarrow A_6) = \text{Primeros}(A_6) = \{d\}.$$

$$SD(A_4 \rightarrow A_5) = \{, b, c, \$\}.$$

Es $\text{primeros}(A_5)$ \cup $\text{Siguientes}(A_4)$.

$$SD(A_3 \rightarrow \lambda) = \text{Siguientes}(A_3) = \{c\}.$$

$$SD(A_3 \rightarrow b) = \{b\}.$$

$$SD(A_1 \rightarrow \lambda) = \text{Siguientes}(A_1) = \{\$\}.$$

$$\begin{aligned} SD(A_1 \rightarrow A_2 A_1) &= \text{Primeros}(A_2) \\ &= \text{Primeros}(A_3) \cup \text{Siguientes}(A_3) \\ &= \{b\} \cup \{c\}. \end{aligned}$$

La derivada es $LL(1)$.

Entonces

$$G_1' = \langle \{S\}, \{a, b, c, d, \lambda\}, S, P_1' \rangle$$

$$\text{con } P_1': S \rightarrow a((b|\lambda)c(c|d)^*|dd^*)a)^*$$

es $ELL(1)$, y vale $L(G_1) = L(G_1')$.

```

b) S() {
    match('a');
    while (tc in {b, c}) {
        if (tc == 'b') {
            match('b');
        }
        match('c');
        if (tc == ',') {
            while (tc == ',') {
                match(',');
                match('c');
            }
        }
        else if (tc == 'd') {
            match('d');
            while (tc == 'd') {
                match('d');
            }
        }
        else error;
        match('a');
    }
    match('$');
    accept;
}

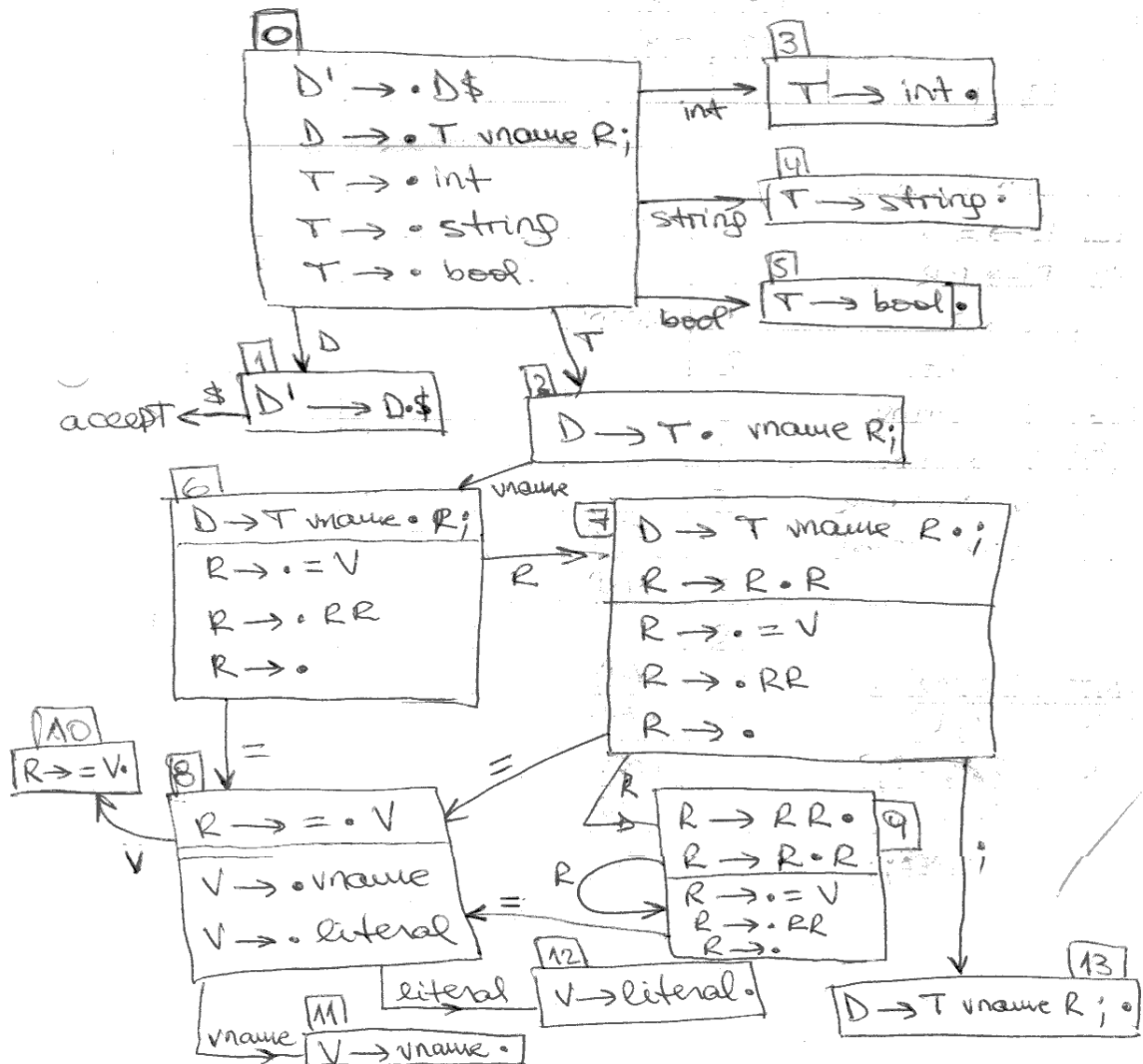
```

Ejercicio 3

$$G_3 = \langle \{D, R, T, V\}, \\ \{vname, ;, =, int, string, bool, literal\}, \\ D, \\ P_3 \rangle$$

$$P_3: \begin{aligned} D &\rightarrow T \text{ vname } R; \\ R &\rightarrow = V \mid R R \mid \lambda \\ T &\rightarrow int \mid string \mid bool \\ V &\rightarrow vname \mid literal. \end{aligned}$$

a) Aumento la gramática con $D' \rightarrow D$ y hago a D' el inicial. Ahora armo el autómata.



La tabla queda:

Estado	Action						
	vname	:	=	int	string	bool	literal
0				s3	s4	s5	
1							
2	s6						
3	r(5)						
4	r(6)						
5	r(7)						
6		r(4)	r(4) s8				
7		s13/r(4)	s8/r(4)				
8	s11	r(3) r(4)	s8/r(3) r(4)				s12
9							
10		r(2)	r(2)				
11		r(8)	r(8)				
12		r(9)	r(9)				
13							

(1) $D \rightarrow T \text{ vname } R ;$							
(2) $R \rightarrow = V$							
(3) $R \rightarrow RR$	0						
(4) $R \rightarrow \lambda$	1	accept					
(5) $T \rightarrow \text{int}$	2						
(6) $T \rightarrow \text{string}$	3						
(7) $T \rightarrow \text{bool}$	4						
(8) $V \rightarrow \text{vname}$	5						
(9) $V \rightarrow \text{literal}$	6						
	7						
	8						
	9						
	10						
	11						
	12						
	13	r(1)					

NT	Siguientes
D	{ \$ }
R	{ = , ; }
T	{ vname }
V	{ = , ; }

b)

 G_3 **no** es SLR dado que tiene conflictos.

Con las siguientes decisiones logramos parsear todas las cadenas posibles sin conflictos

- (6, =): s8
- (7, =): s8
- (7, ;): s13
- (9, =): r3
- (9, ;): r3

La más clara es que en el conflicto s13/r(4) de (7, ;) necesito que esté s13 porque si no nunca proceso ';', que siempre está en las cadenas.

Para las otras decisiones, prestemos atención a la gramática.

$$\begin{aligned}
 D &\rightarrow T \text{ \texttt{vname}} R; \\
 R &\rightarrow = V \mid RR \mid \lambda \\
 T &\rightarrow \texttt{int} \mid \texttt{string} \mid \texttt{bool} \\
 V &\rightarrow \texttt{vname} \mid \texttt{literal}
 \end{aligned}$$

Podríamos decir que el “lenguaje” generado es el mismo que el de la “expresión regular”

$$(\texttt{int} \mid \texttt{string} \mid \texttt{bool}) \texttt{vname} (= V)^* ;$$

(Las comillas se sacarían si V se reemplazara por $(\texttt{vname} \mid \texttt{literal})$, pero para los fines de la explicación es mejor que quede esa expresión, pensemos que es un símbolo terminal.)

Vamos a separar en 3 casos:

- I. $(\texttt{int} \mid \texttt{string} \mid \texttt{bool}) \texttt{vname} ;$
- II. $(\texttt{int} \mid \texttt{string} \mid \texttt{bool}) \texttt{vname} = V ;$
- III. $(\texttt{int} \mid \texttt{string} \mid \texttt{bool}) \texttt{vname} (= V)^k ;$ considerando $k \geq 2$

Sin pérdida de generalidad, vamos a ver que parsea las cadenas con **int**, separando en 3 casos:

I.

Pila	Símbolos	Cadena	Acción
0	\$	int vname ; \$	s3
0 3	\$ int	vname ; \$	r5
0 2	\$ T	vname ; \$	s6
0 2 6	\$ T vname	; \$	r4
0 2 6 7	\$ T vname R	; \$	s13
0 2 6 7 13	\$ T vname R ;	\$	r1
0 1	\$ D	\$	accept

II.

Pila	Símbolos	Cadena	Acción
0	\$	int vname = V ; \$	s3
0 3	\$ int	vname = V ; \$	r5
0 2	\$ T	vname = V ; \$	s6
0 2 6	\$ T vname	= V ; \$	s8
0 2 6 8	\$ T vname =	V ; \$	s10
0 2 6 8 10	\$ T vname = V	; \$	r2
0 2 6 7	\$ T vname R	; \$	s13
0 2 6 7 13	\$ T vname R ;	\$	r1
0 1	\$ D	\$	accept

III.

Pila	Símbolos	Cadena	Acción
0	\$	int vname = V = V ; \$	s3
0 3	\$ int	vname = V = V ; \$	r5
0 2	\$ T	vname = V = V ; \$	s6
0 2 6	\$ T vname	= V = V ; \$	s8
0 2 6 8	\$ T vname =	V = V ; \$	s10
0 2 6 8 10	\$ T vname = V	= V ; \$	r2
0 2 6 7	\$ T vname R	= V ; \$	s8
0 2 6 7 8	\$ T vname R =	V ; \$	s10
0 2 6 7 8 10	\$ T vname R = V	; \$	r2
0 2 6 7 9	\$ T vname R R	; \$	r3
0 2 6 7	\$ T vname R	; \$	s13
0 2 6 7 13	\$ T vname R ;	\$	r1
0 1	\$ D	\$	accept

Pila	Símbolos	Cadena	Acción
0	\$	int vname = V = V = V ; \$	s3
0 3	\$ int	vname = V = V = V ; \$	r5
0 2	\$ T	vname = V = V = V ; \$	s6
0 2 6	\$ T vname	= V = V = V ; \$	s8
0 2 6 8	\$ T vname =	V = V = V ; \$	s10
0 2 6 8 10	\$ T vname = V	= V = V ; \$	r2
0 2 6 7	\$ T vname R	= V = V ; \$	s8
0 2 6 7 8	\$ T vname R =	V = V ; \$	s10
0 2 6 7 8 10	\$ T vname R = V	= V ; \$	r2
0 2 6 7 9	\$ T vname R R	= V ; \$	r3
0 2 6 7	\$ T vname R	= V ; \$	s8
...
0 1	\$ D	\$	accept

Cada = V se consume y se vuelve a esta línea roja con un = V menos que leer en la cadena, así como de la verde se llegó a esta roja. Va a seguir así hasta llegar a lo azul de la tabla anterior.

2017 C2 Parcial

1. (30 pts) Dada la gramática $G_1 = \langle \{S, X, Y, A, B, C\}, \{a, b, c, d, e, f, g\}, P_1, S \rangle$, con P_1 :

$$\begin{aligned} S &\rightarrow XY \mid fSg \\ X &\rightarrow A?B?C^* \\ Y &\rightarrow Yea \mid Yeb \mid e \\ A &\rightarrow a \mid b \\ B &\rightarrow b \mid c \\ C &\rightarrow c \mid d \end{aligned}$$

Mostrar que G_1 no es ELL(1). Dar una gramática ELL(1) que genere $L(G_1)$ y que tenga un solo no terminal.

2. (30 pts) Dada la gramática $G_2 = \langle \{S\}, \{a, b\}, P_2, S \rangle$, con P_2 :

$$S \rightarrow aSb \mid Sb \mid \lambda$$

Decidir si G_2 es SLR. En caso contrario, dar una gramática que sea SLR y genere $L(G_2)$.

3. (40 pts) La siguiente gramática describe el lenguaje de comandos para un robot. $G_3 = \langle \{S, C\}, \{\text{Norte}, \text{Sur}, \text{Este}, \text{Oeste}, \text{num}, (,)\}, P_3, S \rangle$, con P_3 :

$$\begin{aligned} S &\rightarrow CS \mid \lambda \\ C &\rightarrow \text{Norte} \mid \text{Sur} \mid \text{Este} \mid \text{Oeste} \mid \text{num } (S) \end{aligned}$$

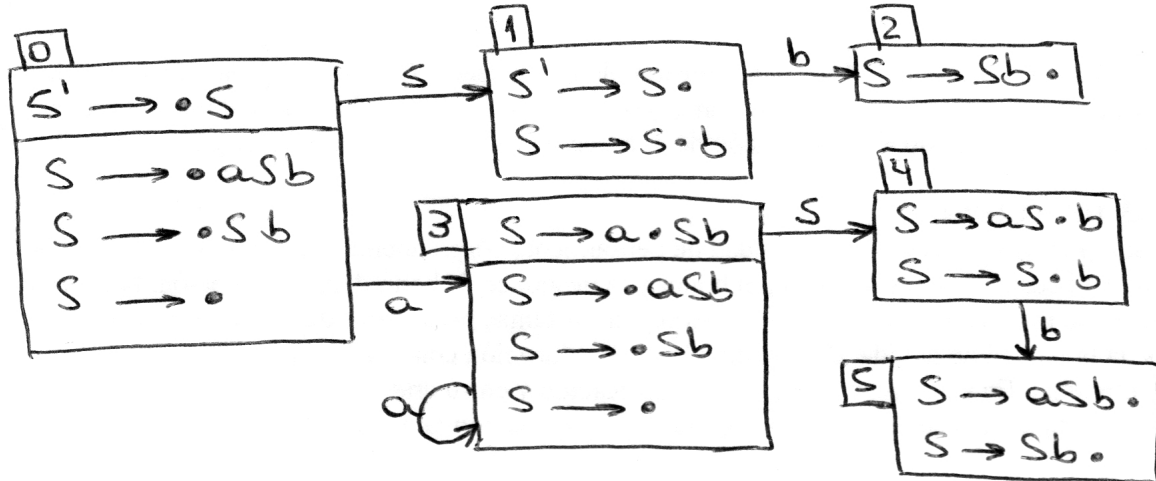
Los comandos que indican un punto cardinal hacen que el robot se desplace una unidad en esa dirección. El terminal `num` representa un número entero y tiene un atributo *valor* de ese tipo. La construcción `num (S)` hace que se repita la secuencia S la cantidad de veces que indique `num.valor`.

Se pide convertir G_3 en una *gramática de atributos* que sintetice en atributos del símbolo inicial la distancia total recorrida y la posición final del robot, asumiendo que parte de las coordenadas (0, 0). Las secuencias que generen giros de 180° deben ser rechazadas.

Por ejemplo, la cadena `Sur Este 3 (Norte Oeste) Sur` es válida, la posición final es (-2, 1), y la distancia recorrida 9 unidades. En cambio, la cadena `Norte Oeste Este` es inválida porque incluye un giro de 180° .

Ejercicio 2

Para decidir si es SLR aumentamos la gramática con la producción $S' \rightarrow S$ y armamos el autómata.



El estado 5 da lugar a dos reducciones distintas para los símbolos terminales de $\text{SIGUIENTES}(S)$, o sea, hay conflictos *reduce-reduce*, y por tanto la gramática no es SLR.

Intentemos darle semántica a los símbolos no terminales y modificar la construcción de las cadenas de modo de incrementar el conjunto de estados, reducir las intersecciones entre los conjuntos SIGUIENTES y disminuir la posibilidad de conflictos en la table de parseo.

Tendremos un terminal E que derivará las cadenas con igual cantidad de as que de bs , y otro B que derivará las que tengan mayor cantidad de bs .

Supongamos que proponemos la siguiente gramática.

$$G_A = \langle \{S, E, B, \tilde{B}\}, \{a, b\}, P_A, S \rangle, \text{ con } P_A$$

$$S \rightarrow E \mid B$$

$$E \rightarrow aEb \mid \lambda$$

$$B \rightarrow E\tilde{B}$$

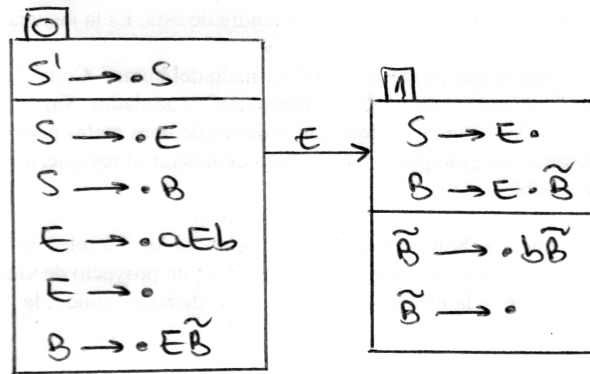
$$\tilde{B} \rightarrow b\tilde{B} \mid \lambda$$

¿Te diste cuenta de que era ambigua? Bueno, como vale la cadena de inclusiones

$$LR(0) \subsetneq SLR \subsetneq LALR(1) \subsetneq LR(1) \subsetneq \text{Gramáticas no ambiguas}$$

seguro esta gramática no va a ser SLR.

De todas formas, empecemos a armar el autómata y veamos qué pasa.



Identificamos que en el estado 1 habrá un conflicto *reduce-reduce* dado que para \$ podrá reducirse por $S \rightarrow E$ y por $\tilde{B} \rightarrow \lambda$. Si prestamos atención, nos damos cuenta de que esto surge justamente de que tenemos dos derivaciones posibles para cadenas con igual cantidad de *as* y *bs*:

$$S \Rightarrow E$$

$$S \Rightarrow B \Rightarrow E\tilde{B} \Rightarrow E\lambda = E$$

Notar que algo fácil de corroborar sobre nuestras gramáticas para detectar algunas fuentes de ambigüedad, aunque no garantiza que no sean ambiguas, es que no existan dos derivaciones $S \rightarrow \alpha$, $S \rightarrow \alpha\beta$, $\alpha, \beta \in (V_N \cup V_T)^*$, con β anulable y ninguna derivación “prefijo” de la segunda.

En nuestro ejemplo, $\alpha = E$, $\beta = \tilde{B}$, β es anulable pues $\tilde{B} \rightarrow \lambda$ es una producción, y la derivación $S \Rightarrow E = \alpha$ no es prefijo de la derivación $S \Rightarrow B \Rightarrow E\tilde{B} = \alpha\beta$. Al existir dos derivaciones que cumplen esas propiedades, nuestra gramática es ambigua.

Los símbolos no terminales de nuestra gramática no tenían la semántica que queríamos pues B no generaba cadenas con una cantidad de *bs* estrictamente mayor que la cantidad de *as*.

Consideremos ahora la gramática corregida y ya aumentada

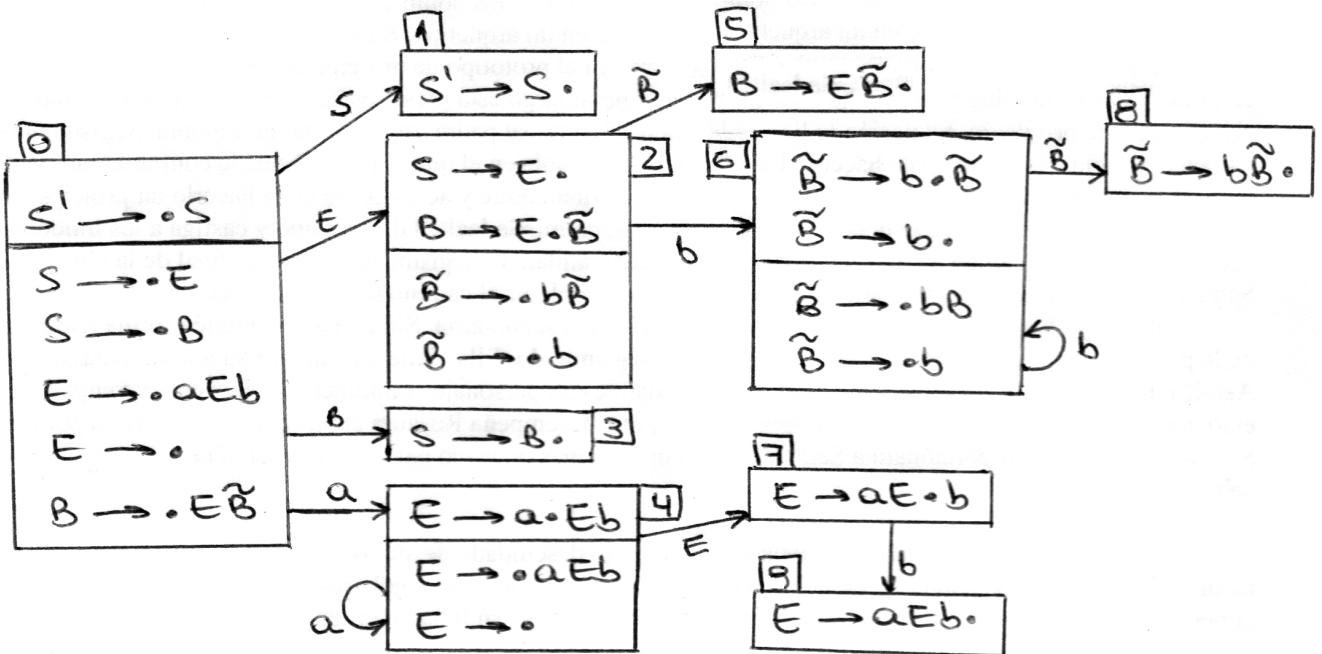
$$G_F = \langle \{S', S, E, B, \tilde{B}\}, \{a, b\}, P_F, S' \rangle, \text{ con } P_F$$

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow E \mid B \\ E &\rightarrow aEb \mid \lambda \\ B &\rightarrow E\tilde{B} \\ \tilde{B} &\rightarrow b\tilde{B} \mid b \end{aligned}$$

SÍMBOLO NO TERMINAL	SIGUIENTES
S'	$\{\$ \}$
S	$\{\$ \}$
E	$\{b, \$ \}$
B	$\{\$ \}$
\tilde{B}	$\{\$ \}$

(Se podría reemplazar la producción $S \rightarrow B$ por $S \rightarrow E\tilde{B}$ y eliminar el símbolo no terminal B con su producción $B \rightarrow E\tilde{B}$.)

El autómata queda



Numeramos las producciones y armamos la tabla de parseo.

- $r1 = r(S \rightarrow E)$
- $r2 = r(S \rightarrow B)$
- $r3 = r(E \rightarrow aEb)$
- $r4 = r(E \rightarrow \lambda)$
- $r5 = r(B \rightarrow E\tilde{B})$
- $r6 = r(\tilde{B} \rightarrow b\tilde{B})$
- $r7 = r(\tilde{B} \rightarrow b)$

ESTADO	ACTION			GoTo			
	a	b	$\$$	S	E	B	\tilde{B}
0	s4	r4	r4	1	2	3	
1			accept				
2		s6	r1				5
3			r2				
4	s4	r4	r4		7		
5			r5				
6		s6	r7				8
7		s9					
8			r6				
9		r3	r3				

Como se puede ver, no hay conflictos y por tanto esta gramática es SLR.

Veamos qué pasa con

$$G'_2 = \langle \{S, E, B\}, \{a, b\}, P'_2, S \rangle, \text{ con } P'_2$$

$$S \rightarrow E \mid B$$

$$E \rightarrow aEb \mid \lambda$$

$$B \rightarrow aBb \mid Bb \mid b$$

Se tendría un conflicto *shift-reduce* porque el estado 0 shiftea con b por $B \rightarrow \cdot b$ y reduce con b por $E \rightarrow \cdot$. Además, después de transicionar con a se llegaría a un estado con $B \rightarrow a \cdot Bb$ en

el *kernel* y $B \rightarrow \cdot Bb$ en la clausura, que eventualmente daría lugar a un conflicto *reduce-reduce* después de transicionar con B y con b .

O sea que como era de esperarse, no es meramente una cuestión de darle semántica a los símbolos no terminales, pero esto es útil para entender la gramática y para evitar ciertas fuentes de ambigüedad (en el caso de G_A , E y B generaban cadenas en común haciendo ambigua la gramática).

Ejercicio 3

Vamos a tener que verificar usando la función `CONDITION` que no se realicen giros de 180° . Para eso, C deberá heredar de S un atributo *últimaDirección* que compare con la dirección que produzca. Suponiendo que no fuera un giro inválido, C debería actualizar su última dirección en base a la dirección del desplazamiento (que vendría de sus hijos y por tanto sería un atributo sintetizado) y pasársela a su hermana S . Como este atributo no puede ser sintetizado y heredado, tendremos otro atributo *direcciónActualizada* que será sintetizado para C y que heredará S en *S.últimaDirección*.

Por otro lado, necesitamos un atributo que sirva para comparar la dirección del último movimiento en la secuencia que deriva S en `num(S)` con la primera de esa secuencia, para el caso de que `num` tenga valor mayor a 1. Este atributo heredado será *direcciónReinicio*, y el valor de `num` estará en el atributo heredado *num*.

También tendremos un atributo sintetizado *distancia* que se calculará en base a las distancias de los nodos hijos.

Por último, nos faltan atributos para las posiciones en x y en y , que también podrán calcularse a partir de los desplazamientos parciales de los nodos hijos, por lo que serán atributos sintetizados.

ATRIBUTOS	TIPO	S o H
<i>S.últimaDirección</i>	Char	H
<i>S.direcciónActualizada</i>	Char	S
<i>S.direcciónReinicio</i>	Char	H
<i>S.x</i>	Int	S
<i>S.y</i>	Int	S
<i>S.distancia</i>	Int	S
<i>S.num</i>	Int	H
<i>C.últimaDirección</i>	Char	H
<i>C.direcciónActualizada</i>	Char	S
<i>C.direcciónReinicio</i>	Char	H
<i>C.reinicioActualizada</i>	Char	S
<i>C.x</i>	Int	S
<i>C.y</i>	Int	S
<i>C.distancia</i>	Int	S
<i>C.num</i>	Int	H

PRODUCCIONES	ACCIONES
$S \rightarrow C S_1$	$\{$ $C.últimaDirección = S.últimaDirección;$ $C.direcciónReinicio = S.direcciónReinicio;$ $S_1.últimaDirección = C.direcciónActualizada;$ $S_1.direcciónReinicio = C.reinicioActualizada;$ $C.num = S.num$ $S_1.num = S.num$ $S.x = C.x + S_1.x;$ $S.y = C.y + S_1.y;$ $S.distancia = C.distancia + S_1.distancia; \}$
$S \rightarrow \lambda$	$\{$ $CONDITION(S.num \geq 2 \Rightarrow$ $((S.direcciónReinicio == 'S' \Rightarrow S.direcciónActualizada \neq 'N')$ $\wedge (S.direcciónReinicio == 'N' \Rightarrow S.direcciónActualizada \neq 'S')$ $\wedge (S.direcciónReinicio == 'E' \Rightarrow S.direcciónActualizada \neq 'O')$ $\wedge (S.direcciónReinicio == 'O' \Rightarrow S.direcciónActualizada \neq 'E')));$ $S.distancia = 0;$ $S.direcciónActualizada = S.últimaDirección;$ $S.x = 0; \quad S.y = 0; \}$
$C \rightarrow Norte$	$\{$ $CONDITION(C.últimaDirección \neq 'S');$ $C.reinicioActualizada = IF(C.direcciónReinicio == null, 'N', C.direcciónReinicio)$ $C.direcciónActualizada = 'N';$ $C.x = 0; \quad C.y = 1;$ $C.distancia = 1; \}$
$C \rightarrow Sur$	$\{$ $CONDITION(C.últimaDirección \neq 'N');$ $C.reinicioActualizada = IF(C.direcciónReinicio == null, 'S', C.direcciónReinicio)$ $C.direcciónActualizada = 'S';$ $C.x = 0; \quad C.y = -1;$ $C.distancia = 1; \}$
$C \rightarrow Este$	$\{$ $CONDITION(C.últimaDirección \neq 'O');$ $C.reinicioActualizada = IF(C.direcciónReinicio == null, 'E', C.direcciónReinicio)$ $C.direcciónActualizada = 'E';$ $C.x = 1; \quad C.y = 0;$ $C.distancia = 1; \}$
$C \rightarrow Oeste$	$\{$ $CONDITION(C.últimaDirección \neq 'E');$ $C.reinicioActualizada = IF(C.direcciónReinicio == null, 'O', C.direcciónReinicio)$ $C.direcciónActualizada = 'O';$ $C.x = -1; \quad C.y = 0;$ $C.distancia = 1; \}$
$C \rightarrow num(S)$	$\{$ $CONDITION(0 \leq num.valor);$ $S.num = num.valor;$ $S.últimaDirección = C.últimaDirección;$ $C.direcciónActualizada = S.direcciónActualizada;$ $C.reinicioActualizada = IF(C.num \neq null, S.direcciónReinicio, null);$ $C.x = num.valor * S.x;$ $C.y = num.valor * S.y;$ $C.distancia = num.valor * S.distancia; \}$

En la primera producción de $S \rightarrow CS_1$, $S.últimaDirección$ tiene valor null, eso se hereda a C , y la comparación en Condition da bien (en caso de que se haga, si no se sigue heredando hasta que llegue donde corresponda).

En la producción $C \rightarrow \text{num}(S)$, $S.direcciónReinicio$ tiene valor null.

La condición de $direcciónReinicio$ se verifica en $S \rightarrow \lambda$ porque si se verificara con alguna producción de C , se debería estar seguro de que esa fuera la última de la secuencia, o sea, la que fuera previa al “reinicio”.

$C.reinicioActualizada = \text{IF}(C.num \neq \text{null}, S.direcciónReinicio, \text{null})$; está por si hay algo como $\text{num}(\text{num}(\text{Este Sur}) \text{ Oeste})$.